Apple Assembly Line

$1.80

Volume 4 -- Issue 10 July, 1984

In This Issue...

Feedback on our DOSonomy

Our little dossier of DOS names was well received. It may be we will soon have so many names we will need a dosser (a large basket that can be carried on the back) to hold them all. On the other hand, if we keep writing about this our fortunes may reverse, forcing to finding new quarters in a doss house. What is the critical dosage?

Dan Pote offers "Kinda-Sorta-DOS". Which led Bill to coin "MaybeDOS". Randy Horton reminded us of "Ante-DiluviDOS". Chris Balthrop enters MacroDOS and "What's Up DOS". (I think the latter is "Buggy". Or "Bugsy"? Oh, it's not bunny anymore...) If you can take all this, you may be too docile.

Don Lancaster Strikes Again

We just have a little space and a little time to mention Don's new Assembly Cookbook for the Apple II/IIe, which just arrived. It looks like another winner! Look for a full review next month, or check our ad on page 3 for ordering info.

18-digit Arithmetic, Part 3.....Bob Sander-Cederlof

Plowing ahead, this installment will offer the division and input conversion subroutines.

You will remember that we covered addition and subtraction in the May 1984 issue, and multiplication in June. Now it's time for division, which completes the fundamental arithmetic operations. All four of these routines are designed to operate on two arguments stored in DAC and ARG, leaving the result in DAC. Addition and subtraction leave "garbage" in ARG. Multiplication leaves ARG unchanged. Division leaves in ARG what was in DAC.

Division is simple enough in concept, but no one would call it simple in implementation. "How many groups of X are in Y?" "If I deal an entire deck of 52 cards to 4 people, how many will each person get?" "If I scramble a dozen eggs, and serve them in equal-size portions to 7 people, how many eggs will each eat?" (Really, I am good cook!)

Suppose I have a pile of pennies, and want to find out how many dollars they represent. I will count out piles of 100 pennies, moving them into separate piles. Then I will count the little piles. Now, suppose I have two 18-digit numbers in my computer and want to divide the one in ARG by the one in DAC.... I will subtract the value in DAC from the one in ARG over and over, until I finally cross zero. Then if I was wise enough to count how many times I did the subtraction, I have the answer.

Let's look at the problem in more detail now. What I want to do is divide the value in ARG by the value in DAC:

$$\frac{\text{numerator (in ARG)}}{\text{denominator (in DAC)}} = \text{quotient (in DAC)}$$

Numbers in DP18 can be positive or negative, so we have to remember the rules of signed division. If the signs of the numerator and denominator are the same, the quotient will be positive; if they are different, the quotient will be negative.

Numbers in DP18 are coded as 18-digit fractions with a power-of-ten exponent. Remembering algebra:

$$\frac{.f * 10^{m}}{.g * 10^{n}} = \frac{f}{g} * 10^{(m-n)}$$

The 18-digit fractions are normalized so that there are no leading zeroes. That is, the value will either be all zero, or it will be between .1 and .999999999999999999 (inclusive).

I think it is time now to start looking at the program. In the listing which follows there are references to subroutines and variables which we defined in the previous two installments of this series.

S-C Macro Assembler Version 1.0.....$80
S-C Macro Assembler Version 1.1.....$92.50
Version 1.1 Update.....$12.50
Source Code for Version 1.1 (on two disk sides).....$100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility (without source code).....$20
S-C Cross Reference Utility (with complete source code).....$50
DISASM Dis-Assembler (RAK-Ware).....$30
Source Code for DISASM.....additional $30

S-C Word Processor (with complete source code).....$50
Double Precision Floating Point for Applesoft (with source code).....$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each $15
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics).....$39
Quick-Trace (Anthro-Digital).....(reg. $50) $45
Visible Computer: 6502 (Software Masters).....(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor.....$60
Amper-Magic (Anthro-Digital).....(reg. $75) $65
Amper-Magic Volume 2 (Anthro-Digital).....(reg. $35) $30
Routine Machine (Southwestern Data Systems).....(reg. $64.95) $60
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36
FLASH! Integer BASIC Compiler (Laumer Research).....$79
Fontrix (Data Transforms).....$75
Aztec C Compiler System (Manx Software).....(reg. $199) $180

Blank Diskettes (Verbatim).....2.50 each, or package of 20 for $45
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each or $25 per 100
These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers..... 6 cents each
ZIP Game Socket Extender (Ohm Electronics) $20

Books, Books, Books.....compare our discount prices!
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
Second edition, with //e information.
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Incredible Secret Money Machine", Lancaster.....($7.95) $7
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Assembly Lines: The Book", Roger Wagner.....($19.95) $18
"What's Where in the Apple", Second Edition.....($19.95) $19
"What's Where Guide" (updates first edition).....($9.95) $9
"6502 Assembly Language Programming", Leventhal.....($18.95) $18
"6502 Subroutines", Leventhal.....($17.95) $17
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

Line 4250 swaps the contents of ARG and DAC. I did it this way because it leaves something possibly useful in ARG after the division is finished. If you wanted to form the reciprocal quotient, DAC=DAC/ARG, you can enter at DDIVR, which skips the swapping step.

Lines 4260-4270 check for the illegal case of division by zero. If I divide something into zero-size parts, I get an infinite number of these parts. That's fine, but the DP18 has no representation for infinity; therefore we say it is illegal to divide by zero, just like Applesoft does. Some computers and some software arithmetic packages do represent infinity, but DP18 does not. Zero values are represented by having an exponent byte of zero, so we only have to check one byte here.

Lines 4280-4310 form the sign of the quotient. This is the same as lines 1280-1310 of the DMULT listing given last month, and so we could make them into a subroutine. The subroutine would take 10 bytes, and the two JSR's make another 6. That's 16 bytes, against the 18 bytes for the two versions of in-line code. Saves a total of 2 bytes, at a cost of adding 12 cpu cycles to both multiply and divide. (Small digression into the kind of trade-offs I am continually making....)

Lines 4330-4390 compute the exponent of the quotient, and check for overflow and underflow cases. The special case of the numerator being zero is also caught here, line 4350. Line 4380 restores the bias of $40. Bias? Remember, the exponent is kept in memory with $40 added to it, so that the range -63 through +63 is represented by $01 through $7F.

If the new exponent is still in the range $00 through $7F, we will go ahead and do the division. If not, the quotient is either too small (underflow) or too large (overflow). For example, 10^-40 / 10^40 results in 10^-80, which is too small for DP18. Lines 4410-4470 catch these cases, and change the quotient to zero. If the new exponent is between $80 and $BF, it represents 10^64 or larger, and so we call on the Applesoft OVERFLOW error.

Lines 4500-4550 set up the loop which does the actual division of the fractions. The 6502's decimal mode will be used during this loop. Ten bytes in MAC (defined in DMULT last month) will be used to hold the quotient until we are through with DAC. The X-register will be used to count out the 20 digits. The other end of the loop is in lines 4920-4930, where X is decremented and tested.

The body of the loop is really a lot simpler than it looks. Basically, ARG is subtracted from DAC until DAC goes negative. The number of subtractions is counted in MAC+9. Then ARG is added back to DAC to make it positive again, and MAC+9 decremented. The result is a quotient digit in MAC+9, and a remainder in DAC. One extra digit is needed, extending DAC on the left end. This digit is carried in the stack. See it pushed at line 4710, pulled at line 4790.

After each digit of the quotient is determined, both MAC and DAC are shifted left one digit place. This might shift a significant digit out of DAC (the remainder), so it is lifted out first and saved on the stack (lines 4570-4630). If the first two digits of the remainder (happen to be "00", then we know without subtracting that the quotient digit in this position will also be "0". (Remember that the leading digit of the denominator in ARG is NEVER zero.) This fact can speed up divisions, so it is tested for at line 4580, with lines 4670-4680.

After all 20 digits are formed, the loop terminates. Line 4950 then returns us to binary mode. Line 4960 adds one to the quotient exponent, adjusting for the normalization step. (.9/.1 = 9, but we want to represent it as .9*10^1.) If the exponent now is negative ($80), it may be still in range if the leading digit of the quotient is zero (.1/.9 = 0.1111...). This test takes place at lines 4970-5000.

Lines 5020-5060 copy the quotient from MAC to DAC. These are the same as lines 1330-1370 in DMULT, so they could be made into a subroutine. Two other candidates for subroutines are lines 4720-4780, which are identical to lines 1680-1740 of DADD (May 1984); and lines 4830-4890, which are the same as 1530-1590 of DADD.

Finally, DDIV exits by jumping to NORMALIZE.DAC.

Doesn't all this take a lot of time? You bet it does! I timed it in the full DP18 package with a program that looked like this:

```
&DP:INPUT X(0) : INPUT X(2)
FOR I = 1 TO 100
&DP:X(4) = X(0)/X(2)
NEXT
```

I determined the loop overhead by entering a value zero for X(0). Since this case skips around nearly everything in DDIV, I called its time the loop overhead time. After subtracting out the loop overhead, the times look like this:

0/anything	0
x/x	12 msec
1/9=.1111...	23 msec
8/9=.8888...	49 msec
1/7=.142857...	35 msec

It looks like the maximum time, which would be for a quotient with all 20 digits = 9, would be about 53 msec. The average time, about 35 msec. This compares with an average Applesoft 9-digit division time of about 7 msec.

```
1000 *SAVE S.DP18 DIVIDE
4220 *--------------------------------
4230 *  DAC = ARG / DAC
4240 *--------------------------------
4250 DDIV   JSR SWAP.ARG.DAC  ...CHANGE TO DAC = DAC/ARG
4260 DDIVR  LDA ARG.EXPONENT  CHECK FOR ZERO DENOMINATOR
4270        BEQ .2            ...X/0 IS ILLEGAL
```

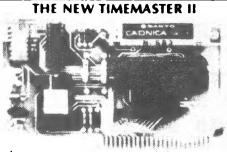
```
4280 *---FORM SIGN OF QUOTIENT--------
4290         LDA DAC.SIGN
4300         EOR ARG.SIGN
4310         STA DAC.SIGN
4320 *---COMPUTE EXPONENT OF QUOTIENT-
4330         SEC
4340         LDA DAC.EXPONENT
4350         BEQ .0          ...0/X=0
4360         SBC ARG.EXPONENT
4370         CLC
4380         ADC #$40        ADJUST OFFSET
4390         STA DAC.EXPONENT
4400 *---CHECK OVER/UNDERFLOW---------
4410         BPL .3          ...NEITHER
4420         ASL             SEE WHICH...
4430         BPL .1          ...OVERFLOW
4440 .0      LDA #0          ...UNDERFLOW, SET RESULT = 0
4450         STA DAC.SIGN
4460         STA DAC.EXPONENT
4470         RTS
4480 .1      JMP AS.OVRFLW
4490 .2      JMP AS.ZRODIV      DIVISION BY ZERO ERROR
4500 *---SET UP QUOTIENT LOOP---------
4510 .3      SED             DECIMAL MODE
4520         LDA #0
4530         STA MAC+9       CLEAR FIRST QUOTIENT DIGIT
4540         LDX #20         DO 20 DIGITS
4550         BNE .5          ...ALWAYS
4560 *---CONTINUE QUOTIENT LOOP-------
4570 .4      LDA DAC.HI
4580         PHP             SAVE ZERO STATUS
4590         LSR
4600         LSR
4610         LSR
4620         LSR
4630         PHA             DAC LEFT EXTENSION
4640         JSR SHIFT.DAC.LEFT.ONE
4650         JSR SHIFT.MAC.LEFT.ONE
4660         PLA             DAC LEFT EXTENSION
4670         PLP             SEE IF FIRST TWO DIGITS = 0
4680         BEQ .9          ...YES, SO QUOTIENT IS ALSO ZERO
4690 *---SUBTRACT WHILE POSSIBLE------
4700 .5      INC MAC+9       COUNT 1 SUBTRACTION
4710         PHA             DAC LEFT EXTENSION
4720         SEC             DO A TRIAL SUB
4730         LDY #9
4740 .7      LDA DAC.HI,Y
4750         SBC ARG.HI,Y
4760         STA DAC.HI,Y
4770         DEY
4780         BPL .7
4790         PLA             DAC LEFT EXTENSION
4800         SBC #0
4810         BCS .5          NO BORROW
4820 *---OVERSHOT, SO RESTORE---------
4830         LDY #9          BORROW,SO ADD IT BACK IN
4840         CLC
4850 .8      LDA DAC.HI,Y
4860         ADC ARG.HI,Y
4870         STA DAC.HI,Y
4880         DEY
4890         BPL .8
4900         DEC MAC+9       BACK OFF QUOTIENT DIGIT, TOO
4910 *---NEXT DIGIT-----------------
4920 .9      DEX             ALL DIGITS?
4930         BNE .4          ...NOT YET, KEEP GOING
4940 *---ADJUST EXP, CHECK OVERFLOW---
4950         CLD             BINARY MODE
4960         INC DAC.EXPONENT  ADJUST FOR OFFSET
4970         BPL .10         ...NO OVERFLOW FROM 64
4980         LDA MAC         COULD BE OVERFLOW
4990         AND #$F0
5000         BNE .1          ...OVERFLOW
```

Apple Peripherals Are All We Make

That's Why We're So Good At It!

THE NEW TIMEMASTER II

Automatically date stamps files with PRO-DOS



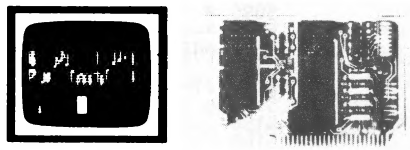
NEW 1984 DESIGN
An official PRO-DOS Clock

- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- NiCad battery keeps the TIMEMASTER II running for over ten years.
- Full emulation of ALL other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the TIMEMASTER to emulate other clocks, we just drop off features. That's why we can emulate others, but others CAN'T emulate us.
- The TIMEMASTER II will automatically emulate the correct clock card for the software you're using. You can also give the TIMEMASTER II a simple command to tell it which clock to emulate (but you'll like the Timemaster mode better). This is great for writing programs for those poor unfortunates that bought some other clock card.
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time (many examples are included).
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.

The TIMEMASTER II includes 2 disks with some really fantastic time oriented programs (over 40) including appointment book so you'll never forget to do anything again. Enter your appointments up to a year in advance then forget them. Plus DOS dater so it will automatically add the date when disk files are created or modified. The disk is over a $200.00 value alone--we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER II. If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER II.

PRICE $129.00

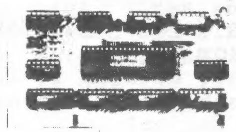
Super Music Synthesizer
Improved Hardware and Software



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all our card's features. Their software sounds the same in our synthesizer.)
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- Runs WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boot.
- A semi-custom I.C. and a low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower case characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	BUILT IN SOFTSWITCH	SHIFT KEY SUPPORT	[illegible]	[illegible]	[illegible]	LIGHT PEN INPUTS	80 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	**179**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
SUPRTERM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	YES	YES
VIEWMAX80	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTERM	MORE	YES	YES	NO	NO	NO	YES	YES	NO
VIDEOTERM	MORE	NO	NO	YES	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $179.00

MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95 K storage in 80 columns! The 64K config. is all that's needed, 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- Precision software disk emulation for Basic, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card, just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster IIe with 128K	**$249**
Upgradeable MemoryMaster IIe with 64K	**$169**
Non-Upgradeable MemoryMaster IIe with 64K	**$149**

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE IIE, II, II+ and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple; A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

Texas Residents Add 5% Sales Tax
Add $10.00 If Outside U.S.A.
Dealer Inquiries Welcome

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 798
Carrollton, TX 75006

Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards

```
5010 *---COPY QUOTIENT TO DAC---------
5020 .10    LDY #9
5030 .11    LDA MAC,Y
5040        STA DAC.HI,Y
5050        DEY
5060        BPL .11
5070        JMP NORMALIZE.DAC
5080 *--------------------------------
5090 *   SHIFT 20 DIGITS IN MAC LEFT ONE PLACE
5100 *--------------------------------
5110 SHIFT.MAC.LEFT.ONE
5120        LDY #4
5130 .1     ASL MAC+9
5140        ROL MAC+8
5150        ROL MAC+7
5160        ROL MAC+6
5170        ROL MAC+5
5180        ROL MAC+4
5190        ROL MAC+3
5200        ROL MAC+2
5210        ROL MAC+1
5220        ROL MAC
5230        DEY
5240        BNE .1
5250        RTS
5260 *--------------------------------
```

DP18 Input Conversion

The input conversion subroutine processes characters from memory to produce a value in DAC. This is analogous to what the equivalent subroutine in Applesoft ROMs does.

It is so analogous, in fact, that I even depend upon CHRGET and CHRGOT to fetch successive characters from memory. It is a lot faster than Applesoft conversion, however, because it is BCD coded rather than binary. This means that, stripping away the frills such as sign, exponent part, and decimal point, it even easier than an ASCII to hex conversion.

Of course, we need all those frills. Look ahead to the program listing which follows: Lines 1200-1220, just those three little lines, handle the conversion of digits. All the rest of the page is for frills! Well, to be honest about it, two of the three lines call subroutines, but still, the frills predominate.

The acceptable format of numbers is basically the same as that which normal Applesoft accepts. A leading sign is optional. The numeric portion can be more than 20 digits long, but only the first 20 will be accumulated (not counting leading zeroes). A decimal point is optional anywhere in the numeric portion. An exponent part can be appended to the numeric portion, and consists of the letter "E", and optional sign, and one or two digits. The exponent can be up to 81, just so the final number evaluates between .1*10^-63 and .9999...9*10^63. Numbers smaller than .1*10^-63 will be changed to zero, and numbers larger than .9999...9*10^63 will cause an OVERFLOW ERROR.

Looking at the program, lines 1040-1080 clear a working area which comprises DAC and four other variables: SGNEXP, EXP, DGTCNT, and DECFLG. SGNEXP will be used to hold the sign of the exponent part; EXP will hold the value of the exponent part; DGTCNT will count the digits in the numeric portion; and

DECFLG will flag the occurrence of a decimal point. DAC includes DAC.SIGN. Note that the X-register will be left with $FF, which fact is important at line 1170 below.

Lines 1090-1100 preset the DAC.EXPONENT to $40, which indicates 10^0. This will be incremented along with DGTCNT until a decimal point is encountered.

Lines 1110-1180 handle the optional leading sign. DAC.SIGN has already been cleared above, indicating the positive case. If a minus sign is in front of the number, line 1170 sets DAC.SIGN negative. Note that calling CHRGOT and CHRGET to retrieve characters automatically eliminates (ignores) blanks. CHRGOT/CHRGET also checks whether the character retrieved is a digit or not, and indicates digits by carry clear. If the first non-blank character is a digit, we immediately jump to the numeric loop at line 1200. If not, the subroutine FIN.SIGN checks for a + or - character. The + or - may or may not be tokenized, depending on whether the string is from an INPUT statement or is a constant embedded in a program, so we have to check for both the character and the token form of both signs. FIN.SIGN handles this checking.

If that first character is neither a digit nor a sign, it may be a letter "E" or a decimal point; so, we go down to lines 1240-1270 to check for those two cases. If neither of these either, we must be at the end of the number. If it is a decimal point, lines 1630-1650 record the fact that a decimal point was found and also check whether this is the first one found or not. If the first, back we go to continue looking for digits. If not the first, it must be the end of the number, so we fall into the final processing section at line 1670.

Exponents are more difficult, because the value actually must be converted from ASCII to binary. Lines 1290-1610 do the work, including handling of the optional sign, and range checking.

Lines 1670-1730 compute the final exponent value. This is the number of digits before the decimal point (not counting any leading zeroes you may have typed to confuse me) plus the exponent computed in the optional "E" field. If the result is negative, between $C0 and $FF, it indicates underflow; in this case, the value is changed to zero. If there were no non-zero digits in the numeric portion, the value is set to zero regardless of any "E" field. If the resulting exponent is between $80 and $BF, it indicates OVERFLOW.

Lines 1840-2130 accumulate individual digits. DGTCNT is used to index into the nybbles of DAC, and the digit is stored directly into place. Leading zeroes on the numeric field are handled here (lines 2090-2120). Leading zeroes before a decimal point are entirely ignored, while leading zeroes after a decimal point cause the DAC.EXPONENT to be decremented. The incrementation of DAC.EXPONENT for each significant digit on the left of the decimal point is also taken care of here (lines 2020-2070).

This complete the third installment of DP18. We are well on the way to a working subset of the entire package. We still need output conversion and some sort of linkage to Applesoft before we can begin to see it all run. The entire DP18 package really exists, and works, now. It includes PRINT USING, very fancy input screen handling, full expression parsing, and all the math functions. Several of you have been very anxious to get the whole package for use in projects of your own, so we have offered a source code license to DP18 on an "as is" basis for only $200.

```
1000 *SAVE S.DP18 FIN
1010 *--------------------------------
1020 *   DP18 INPUT CONVERSION
1030 *--------------------------------
1040 FIN     LDA #0        CLEAR WORK AREA
1050         LDX #WRKSZ-1       (DAC, SGNEXP, EXP,
1060 .1      STA WORK,X          DGTCNT, & DECFLG)
1070         DEX
1080         BPL .1        LEAVE X=$FF WHEN FINISHED
1090         LDA #$40
1100         STA DAC.EXPONENT
1110 *---HANDLE LEADING SIGN----------
1120         JSR AS.CHRGOT
1130         BCC .2        IF DIGIT 0-9
1140         JSR FIN.SIGN  ...SEE IF + OR - SIGN
1150         BNE .4        ...NEITHER + NOR -
1160         BCC .3        ...+
1170         STX DAC.SIGN  ...-, SET TO $FF
1180         BCS .3        ...ALWAYS
1190 *---GET DIGITS TILL NON-DIGIT----
1200 .2      JSR ACCUMULATE.DIGIT
1210 .3      JSR AS.CHRGET   GET NEXT CHARACTER
1220         BCC .2        ...DIGIT
1230 *---".", "E", OR END-------------
1240 .4      CMP #'.       DECIMAL POINT?
1250         BEQ .9        YES
1260         CMP #'E       LETTER E
1270         BNE .10       END OF NUMBER
1280 *---HANDLE EXPONENT FIELD--------
1290         JSR AS.CHRGET
1300         BCC .6        ...DIGIT, ASSUME POSITIVE
1310         JSR FIN.SIGN  ...SEE IF + OR - SIGN
1320         BNE .8        ...NEITHER + NOR -
1330         BCC .5        ...+
1340         ROR SGNEXP    ...-, SO SET SGNEXP NEGATIVE
1350 .5      JSR AS.CHRGET   GET FIRST DIGIT OF EXP
1360         BCS .8          ...NO DIGITS!
1370 .6      AND #$0F        ...ISOLATE EXP 1ST DIGIT
1380         STA EXP
1390         JSR AS.CHRGET   GET 2ND DIGIT OF EXP, IF ANY
1400         BCS .8          ...NO MORE DIGITS
1410         AND #$0F        ISOLATE 2ND DIGIT
1420         PHA             SAVE ON STACK
1430         LDA EXP         MULTIPLY 1ST DIGIT BY 10
1440         ASL
1450         ASL           (CLEARS CARRY TOO)
1460         ADC EXP         #5
1470         ASL             #10  (CARRY STILL CLEAR)
1480         STA EXP         ADD 2ND DIGIT
1490         PLA
1500         ADC EXP
1510         STA EXP         2 DIGIT EXP
1520         CMP #64+18      ALLOW .00000000000000001E+82
1530         BCS .7          OR 99999999999999999E-82
1540         JSR AS.CHRGET   GET NEXT CHAR
1550         BCS .8        NO MORE DIGITS
1560 .7      JMP AS.OVRFLW   OVERFLOW ERROR
1570 .8      ASL SGNEXP    CHECK SIGN OF EXP
1580         BCC .10       ...POSITIVE
```

```
1590          LDA #0          ...NEGATIVE, SO COMPLEMENT EXP
1600          SBC EXP
1610          JMP .11         ...ALWAYS
1620 *---FOUND DECIMAL POINT----------
1630 .9       ROR DECFLG      SET DECIMAL POINT FLAG
1640          BIT DECFLG      CHECK FOR TWO DECIMAL POINTS
1650          BVC .3          NO
1660 *---COMPUTE FINAL EXPONENT-------
1670 .10      LDA EXP         GET EXPLICIT EXPONENT
1680 .11      CLC
1690          ADC DAC.EXPONENT
1700          LDX DGTCNT      SEE IF ANY SIGNIFICANT DIGITS
1710          BNE .12         ...YES
1720          TXA             ...NO, MAKE EXPONENT ZERO
1730 .12      STA DAC.EXPONENT
1740          TAX             TEST RANGE OF EXPONENT
1750          BMI .13         ...NOT IN RANGE 0...7F
1760          RTS
1770 *---EITHER UNDER- OR OVER-FLOW---
1780 .13      ASL             UNDER, OR OVER?
1790          BCC .7          ...OVERFLOW
1800          LDA #0
1810          STA DAC.SIGN
1820          BEQ .12         ...ALWAYS
1830 *-------------------------------
1840 ACCUMULATE.DIGIT
1850          AND #$0F        ISOLATE DIGIT
1860          BEQ .4          ZERO DIGIT
1870          TAX             SAVE DIGIT IN X-REG
1880          LDA DGTCNT      NO MORE THAN 20 SIGNIFICANT DIGITS
1890          CMP #20
1900          BCS .2          DISCARD EXTRA DIGITS
1910 *---STORE THE DIGIT IN DAC-------
1920          LSR             ODD/EVEN TO CARRY
1930          TAY             INDEX TO Y-REG
1940          TXA             GET DIGIT FROM X-REG
1950          BCS .1          ODD DIGIT ON RIGHT SIDE
1960          ASL             EVEN DIGIT MUST BE SHIFTED
1970          ASL
1980          ASL
1990          ASL
2000 .1       ORA DAC.HI,Y MERGE
2010          STA DAC.HI,Y
2020 *---COUNT THE DIGIT--------------
2030 .2       INC DGTCNT      COUNT SIGNIFICANT DIGIT
2040          LDA DECFLG      SEE IF IN FRACTION
2050          BMI .3          YES
2060          INC DAC.EXPONENT  NO
2070 .3       RTS
2080 *---DIGIT = 0--------------------
2090 .4       LDA DGTCNT      SEE IF LEADING ZERO
2100          BNE .2          NO
2110          LDA DECFLG      SEE IF PART OF FRACTION
2120          BPL .5          NO, COMPLETELY IGNORE IT
2130          DEC DAC.EXPONENT
2140 .5       RTS
2150 *-------------------------------
2160 *      SCAN + OR - SIGN
2170 *    --------------------
2180 *       +      .EQ., .CC.
2190 *       -      .EQ., .CS.
2200 *     OTHER    .NE.
2210 *-------------------------------
2220 FIN.SIGN
2230          CMP #'-'
2240          BEQ .2
2250          CMP #TKN.MINUS
2260          BEQ .2
2270          CMP #'+'
2280          BEQ .1
2290          CMP #TKN.PLUS
2300 .1       CLC
2310 .2       RTS
```

Building Label Tables for DISASM.....Bob Sander-Cederlof

RAK-Ware's DISASM has the nice feature of being able to used a list of pre-defined labels when you are disassembling a block of code. I needed to turn the //c monitor ROM ($F800-$FFFF) into source code, and Apple sent me a list of all their labels in this area.

The format of the label table, or name table, is very simple. Each entry takes eight bytes: the first two are the value, high byte first; the remaining six are the label name, in ASCII with high bit set. If the name is less than six characters long, zeroes are used to fill out the entry.

Very simple to explain, but how do you enter things like that in the S-C Macro Assembler? The example on the DISASM disk does it this way:

```
1000      .HS FDED
1010      .AS -/COUT/
1020      .HS 00000000
1030      .HS FDF0
1040      .AS -/COUT1/
1050      .HS 000000
and so on.
```

That works, but it is so error prone and time wasting that I gave up before I started. However, there is an easy way using macros and abbreviations.

Start by defining a macro which will build one entry:

```
1000    .MA LBL
1010    .HS ]1
1020    .AS -/]2/
1030    .BS *+7/8*8-*
1040    .EM
```

The macro is named LBL, and will be used like this:

```
1050    >LBL FDED,COUT
1060    >LBL FDF0,COUT1
```

Line 1030 is the tricky one. This .BS will add just enough zeroes to an entry to pad it out to an even multiple of 8 bytes. Now, assuming the origin started at an even multiple of 8, and assuming you are writing the table on a target file, that macro builds the kind of entries DISASM wants. Instead of just assuming, lets add:

```
0900    .OR $4000
0910    .TF B.NAMETBL
```

I also mentioned abbreviations above. I even get tired of typing "tab>LBL ", you know. Usually when I have a lot of lines to type that have a common element, I use some special character that is easy to type and not present in the lines I plan to type. Then after all the lines are in, I use the

REPLACE command to substitute the longer string for the single-character abbreviation I have used. Thus, I can type:

```
1050 .FDED,COUT
1060 .FDF0,COUT1
et cetera
```

and after many lines type

```
REP /./ >LBL /1050,A
```

I was about up to FA90 when it dawned on me that I could break the symbols into blocks within a page, and include the page value in my abbreviation:

```
1050 .ED,COUT
1060 .F0,COUT1
REP /./ >LBL FD/1050,A
```

With all these shortcuts, I was able to enter over 400 label names and definitions in less than an hour.

Let the computer work FOR you!

Quick Memory Testing.....Bob Sander-Cederlof

What do you do when a friend brings his Apple over with an intermittent memory failure? You KNOW you have a memory test program somewhere, but WHERE?

Here is a quick way to test normal RAM between $7D0 and $BFFF. (RAM in //e hyperspace or banked into ROM space is another matter.) Turn on your friend's computer, and hit reset to abort the booting sequence. We don't need or want DOS around while we are testing memory. Type HOME and CALL-151 to get into the monitor. Then type the following monitor command:

```
*N 7D0:00 N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   7D0:55 N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   7D0:AA N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   7D0:FF N 7D1<7D0.BFFEM 7D1<7D0.BFFEV
   34:0
```

The "*" is the monitor prompt, so don't you type it. There are no carriage returns in the line above, it just wraps around the 40-column screen that way. There must be one trailing blank after the "34:0" at the end. This makes the monitor repeat the whole command line forever.

I started the test at $7D0 so there will be some visible feedback, but most of the screen will stay clear. If you begin testing at a lower address, any errors displayed on the screen might be overwritten as soon as they show up.

When you type the RETURN key you will see a line of inverse at-signs at the bottom of the screen. After a few seconds, this will change to flashing U. Then *, and then some other character, depending on what kind of Apple you have. Then the cycle will start over again.

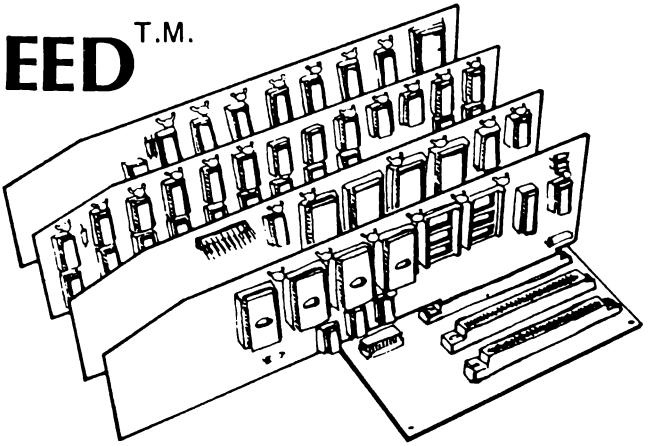
Until a memory error is detected. Any error will cause two lines to be printed, giving the address before the error with its contents and the contents of the error byte, and the address of the error byte with its actual contents and should-be contents. For example, if you were in the "AA" phase, and $8123 came up with $AB, you would see:

```
8122-AA (AB)    byte before error
8123-AB (AA)    error byte
```

If any error lines start printing, note which bit is bad and which 16K bank it is in. Then you can point directly to the bad chip.

	7	6	5	4	3	2	1	0
7D0...3FFF	C10	C9	C8	C7	C6	C5	C4	C3
4000...7FFF	D10	D9	D8	D7	D6	D5	D4	D3
8000...BFFF	E10	E9	E8	E7	E6	E5	E4	E3

APPLESEED T.M.



Appleseed is a complete computer system. It is designed using the bus conventions established by Apple Computer for the Apple][. Appleseed is designed as an alternative to using a full Apple][computer system. The Appleseed product line includes more than a dozen items including CPU, RAM, EPROM, UART, UNIVERSAL Boards as well as a number of other compatible items. This ad will highlight the Mother board.

BX-DE-12 MOTHER BOARD

The BX-DE-12 Mother board is designed to be fully compatible with all of the Apple conventions. Ten card slots are provided. Seven of the slots are numbered in conformance with Apple standards. The additional three slots, lettered A, B and C, are used for boards which don't require a specific slot number. The CPU, RAM and EPROM boards are often placed in the slots A, B and C.

The main emphasis of the Appleseed system is illustrated by the Mother Board. The absolute minimum amount of circuitry is placed on the Mother Board; only the four ICs which are required for card slot selection are on the mother board. This approach helps in packaging (flexibility & smaller size), cost (buy only what you need) and repairability (isolate and fix problems through board substitution).

Appleseed products are made for O.E.M.s and serious industrial/scientific users. Send for literature on the full line of Appleseed products; and, watch here, each month, for additional items in the Appleseed line.

Appleseed products are not sold through computer stores.

Order direct from our plant in California.

Apple is a registered trademark of Apple Computer, Inc.

DOUGLAS ELECTRONICS
718 Marina Blvd., San Leandro, CA 94577 • (415) 483-8770

68000 Sieve Benchmark.....Peter J. McInerney
New Zealand

Here are two versions of the Sieve of Eratosthenes for the MC68000. They provide ample justification for the power claimed for this chip.

The first version is a fairly straightforward translation of the algorithm as presented in the November 1982 AAL, by Tony Brightwell. Tony's best time in the 6502 was 183 seconds for 1000 repetitions; in my 12.5 MHz DTACK GROUNDED attached processor, 1000 repetitions took only 40 seconds.

Compare the 68000 code with the 6502 code, and I'm sure you will agree the 68000 version is much easier to understand. Note the use of long instructions in the array clearing loop and the two-dimensional indexing in lines 1230 and 1310. Other nice things are the shift left by 3 (multiply by 8) in line 1270 and the decrement & branch instructions in lines 1120 and 1400. Also very useful is the postincrement address mode, which automatically increments the address kept in the referenced register by 1, 2, or 4 depending on the size of the operation. This is used for popping off (downward growing) stacks or as here to advance through memory. There is also a predecrement mode but I did not use it in these example programs.

The second version uses a modified algorithm. The changes I made should apply to the 6502 version also, improving it in about the same proportion.

* Since we are ignoring even numbers, we may as well leave them out of the array entirely, thus halving the array size.

* We can therefore simplify the formula for odd squares from S*8+1 to S*4.

* We can even do away with the *4 part by adding 4 each time rather than 1.

* The initial array clearing loop can be made faster by using more than one CLR instruction per loop.

This modified version does 1000 iterations in only 33 seconds! It is only slightly harder to follow than the first version, and only slightly larger. In fact, if we forego the final modification above, the code is actually shorter. I think most of the speedup comes from halving the array size.

If you have a Macintosh, and can manage to load machine code into it, you should find everything running about half as fast as my DTACK GROUNDED board.

[We tried the program on our QWERTY Q-68 board, and it took roughly 10 times as long as Peter's DG board. Understandable, since it was using Apple memory at .5MHz rate for all work. (Bill&Bob)]

```
                        1000 *SAVE SIEVE OF ERATOSTHENES.1
                        1010 *-------------------------------
                        1020 *   CODED BY PETER J. MCINERNEY, NEW ZEALAND
                        1030 *-------------------------------
                        1040         .OR $3800
00004000-               1050 ARRAY   .EQ $4000
                        1060 *-------------------------------
00003800- 3C3C 03E7     1070 SIEVE   MOVE    #999,D6    DO 1000 TIMES
                        1080 *---CLEAR WORKING ARRAY----------
00003804- 307C 4000     1090 .1      MOVE    #ARRAY,A0  CLEAR ARRAY FROM
00003808- 303C 0FFF     1100         MOVE    #$FFF,D0   $4000 TO $7FFF
0000380C- 4298          1110 .2      CLR.L   (A0)+
0000380E- 51C8 FFFC     1120         DBF     D0,.2
                        1130 *---INIT VARIABLES---------------
00003812- 7003          1140         MOVEQ   #3,D0      START AT 3
00003814- 7201          1150         MOVEQ   #1,D1      SUM OF ODD NUMBERS
00003816- 7401          1160         MOVEQ   #1,D2      COUNT OF ODD NUMBERS
00003818- 7601          1170         MOVEQ   #1,D3      USED FOR STRIKING NON-PRIMES
0000381A- 307C 4000     1180         MOVE    #ARRAY,A0  START OF ARRAY
0000381E- 6004          1190         BRA.S   .4         JUMP INTO LOOP
                        1200 *---START SIFTING----------------
00003820- 5242          1210 .3      ADDQ    #1,D2      COUNT ODD NUMBERS
00003822- D242          1220         ADD     D2,D1      GET SUM OF ODDS
00003824- 0C30 0000
00003828- 0000          1230 .4      CMPI.B  #0,0(A0,D0) IS THIS A PRIME?
0000382A- 6616          1240         BNE.S   .6           NO
                        1250 *---STRIKE OUT MULTIPLES---------
0000382C- 3801          1260         MOVE    D1,D4      GET 8*S+1 = N*N
0000382E- E744          1270         ASL     #3,D4
00003830- 5244          1280         ADDQ    #1,D4
00003832- 3A00          1290         MOVE    D0,D5      ONLY STRIKE ODD MULTIPLES
00003834- E345          1300         ASL     #1,D5
00003836- 1183 4000     1310 .5      MOVE.B  D3,0(A0,D4)  STRIKE ONE
0000383A- D845          1320         ADD     D5,D4        NEXT STRIKE
0000383C- 0C44 4000     1330         CMPI    #$4000,D4    ...FINISHED?
00003840- 63F4          1340         BLS     .5           ...NO
                        1350 *---GET NEXT SIEVE SIZE----------
00003842- 5440          1360 .6      ADDQ    #2,D0      NEXT ODD NUMBER
00003844- 0C40 007F     1370         CMPI    #127,D0    UNTIL SQRT $4000-1
00003848- 63D6          1380         BLS     .3
                        1390 *---DO IT ALL 1000 TIMES---------
0000384A- 51CE FFB8     1400         DBF     D6,.1      NEXT TIME
0000384E- 4E75          1410         RTS
```

```
                        1000 *SAVE SIEVE OF ERATOSTHENES.2
                        1010 *-------------------------------
                        1020 *   CODED BY PETER J. MCINERNEY, NEW ZEALAND
                        1030 *-------------------------------
                        1040         .OR $3800
00004000-               1050 ARRAY   .EQ $4000
                        1060 *-------------------------------
00003800- 3C3C 03E7     1070 SIEVE   MOVE    #999,D6    DO 1000 TIMES
                        1080 *---CLEAR WORKING ARRAY----------
00003804- 307C 4000     1090 .1      MOVE    #ARRAY,A0  CLEAR ARRAY FROM
00003808- 303C 00FF     1100         MOVE    #$FF,D0    $4000 TO $7FFF
0000380C- 4298          1110 .2      CLR.L   (A0)+
0000380E- 4298          1120         CLR.L   (A0)+
00003810- 4298          1130         CLR.L   (A0)+
00003812- 4298          1140         CLR.L   (A0)+
00003814- 4298          1150         CLR.L   (A0)+
00003816- 4298          1160         CLR.L   (A0)+
00003818- 4298          1170         CLR.L   (A0)+
0000381A- 4298          1180         CLR.L   (A0)+
0000381C- 51C8 FFEE     1190         DBF     D0,.2
                        1200 *---INIT VARIABLES---------------
00003820- 7003          1210         MOVEQ   #3,D0      START AT 3
00003822- 7804          1220         MOVEQ   #4,D4      CORRESPONDS TO 9
00003824- 7404          1230         MOVEQ   #4,D2      DELTA
00003826- 7601          1240         MOVEQ   #1,D3      USED FOR STRIKING NON-PRIMES
00003828- 307C 4001     1250         MOVE    #ARRAY+1,A0 POSITION OF 9
0000382C- 327C 4000     1260         MOVE    #ARRAY,A1  START OF ARRAY
00003830- 6004          1270         BRA.S   .4         JUMP INTO LOOP
                        1280 *---START SIFTING----------------
00003832- 5842          1290 .3      ADDQ    #4,D2      UPDATE DIFFERENCE
00003834- D842          1300         ADD     D2,D4      UPDATE SQUARE POINTER
00003836- 0C18 0000     1310 .4      CMPI.B  #0,(A0)+   IS THIS A PRIME?
0000383A- 660E          1320         BNE.S   .6           NO
                        1330 *---STRIKE OUT MULTIPLES---------
0000383C- 3A04          1340         MOVE    D4,D5      GET LATEST SQUARE
0000383E- 1383 5000     1350 .5      MOVE.B  D3,0(A1,D5)  STRIKE ONE
00003842- DA40          1360         ADD     D0,D5        NEXT MULTIPLE
00003844- 0C45 2000     1370         CMPI    #$2000,D5    ...FINISHED?
00003848- 63F4          1380         BLS     .5           ...NO
                        1390 *---GET NEXT SIEVE SIZE----------
0000384A- 5440          1400 .6      ADDQ    #2,D0      NEXT ODD NUMBER
0000384C- 0C40 007F     1410         CMPI    #127,D0    UNTIL SQRT $4000-1
00003850- 63E0          1420         BLS     .3
                        1430 *---DO IT ALL 1000 TIMES---------
00003852- 51CE FFB0     1440         DBF     D6,.1      NEXT TIME
00003856- 4E75          1450         RTS
```

Updating the 6502 Prime Sifter.....Bob Sander-Cederlof

I spent a half day applying Peter's algorithm improvements to the November 1982 6502 version, and refining the program as much as I could. It now runs in 175 milliseconds per iteration, or 1000 iterations in 175 seconds. Still way behind the 68000, of course. On the other hand, a 6MHz 6502, with fast enough RAM for no wait states, would be faster than a 12.5 MHz 68000. And it remains to be seen what the 65802 could do.

In the process of running various versions and various tests, I discovered that the innermost loop, at lines 1820-1850, is executed 10277 times. This means that, while marking out the odd non-primes between 1 and 16383, a total of 10277 such marks are made. Since only odd numbers are assigned slots in the working array, giving only 8192 such slots, you can see that some numbers get stricken more than once. These are the numbers with more than one prime factor. The most-stricken number is 3*5*7*11*13 = 15015, which gets five strikes. The loop takes 11 cycles as written, and I don't see any way to shorten it any further or to reduce the number of times it is used. Do you?

The loop time is 11*10277 = 121297 cycles, or about 120 msec out of the total 175. The array clearing accounts for another 41 msecs, leaving only 14 msec for all the rest of the program. Not bad!

Here is a little Applesoft program which will make a nice neat listing of primes from the working array, assuming it runs from $6000 through $7FFF.

```
100 HIMEM:24576
110 FOR A = 24576 TO 32767
120 IF PEEK (A) = 0 THEN
    PRINT RIGHT$("      "+STR$((A-24576)*2+1,7);:
    N = N + 1
130 IF N = 10 THEN PRINT : N = 0
140 NEXT
```

```
                1010 *SAVE S.SUPER-FAST PRIMES IMPROVED
                1020         .OR $8000     SAFELY OUT OF WAY
                1030 *--------------------------------
 6000-          1040 BASE    .EQ $6000     BASE OF PRIME ARRAY
 FF3A-          1050 BEEP    .EQ $FF3A     BEEP THE SPEAKER
 00-            1060 SQZZZZ  .EQ 0,1
 02-            1070 START   .EQ 2
 04-            1080 COUNT   .EQ 4,5
                1090 *--------------------------------
                1100         .MA ZERO
                1110         STA ]1+$000,X
                1120         STA ]1+$100,X
                1130         STA ]1+$200,X
                1140         STA ]1+$300,X
                1150         STA ]1+$400,X
                1160         STA ]1+$500,X
                1170         STA ]1+$600,X
                1180         STA ]1+$700,X
                1190         .EM
                1200 *--------------------------------
                1210 *       MAIN CALLING ROUTINE
                1220 *
```

```
8000- A9 9C     1230 MAIN   LDA #-100   DO 1000 TIMES SO WE CAN MEASURE
8002- 85 04     1240        STA COUNT    THE TIME IT TAKES
8004- A9 FF     1250        LDA /-100
8006- 85 05     1260        STA COUNT+1
8008- 20 3A FF  1270        JSR BEEP     ANNOUNCE START
800B- 20 19 80  1280 .1     JSR PRIME
800E- E6 04     1290        INC COUNT
8010- D0 F9     1300        BNE .1
8012- E6 05     1310        INC COUNT+1
8014- D0 F5     1320        BNE .1
8016- 4C 3A FF  1330        JMP BEEP     SAY WE'RE DONE
                1340 *-------------------------------
                1350 *       PRIME ROUTINE
                1360 *       SETS ARRAY STARTING AT BASE
                1370 *       TO $FF IF NUMBER IS NOT PRIME
                1380 *       CHECKS ONLY ODD NUMBERS > 3
                1390 *       INC = INCREMENT OF KNOCKOUT
                1400 *       N = KNOCKOUT VARIABLE
                1410 *-------------------------------
                1420 PRIME
8019- A2 00     1430        LDX #0
801B- 8A        1440        TXA          CLEAR WORKING ARRAY
801C-           1450 .1     >ZERO BASE
8034-           1460        >ZERO BASE+$0800
804C-           1470        >ZERO BASE+$1000
8064-           1480        >ZERO BASE+$1800
807C- E8        1490        INX
807D- D0 9D     1500        BNE .1       NOT FINISHED CLEARING
                1530 *-------------------------------
807F- A9 60     1540        LDA /BASE+4  POINT AT FIRST PRIME-SQUARED
8081- 85 01     1550        STA SQZZZZ+1       (WHICH IS 3*3=9)
8083- A9 04     1560        LDA #BASE+4
8085- 85 00     1570        STA SQZZZZ
8087- A9 01     1580        LDA #1       POINT AT FIRST PRIME (3)
8089- D0 11     1590        BNE .4       ...ALWAYS
                1600 *-------------------------------
808B- 8A        1610 .2     TXA
808C- 0A        1620        ASL
808D- 0A        1630        ASL
808E- 65 00     1640        ADC SQZZZZ
8090- 85 00     1650        STA SQZZZZ
8092- 90 02     1660        BCC .3
8094- E6 01     1670        INC SQZZZZ+1
8096- BD 00 60  1680 .3     LDA BASE,X   GET A POSSIBLE PRIME
8099- D0 23     1690        BNE .8       THIS ONE HAS BEEN KNOCKED OUT
809B- 8A        1700        TXA
                1710 *-------------------------------
809C- 85 02     1720 .4     STA START
809E- 0A        1730        ASL          INC = START*2 + 1
809F- 69 01     1740        ADC #1
80A1- 8D B2 80  1750        STA .7+1
80A4- A5 01     1760        LDA SQZZZZ+1 MOVE MULT TO N
80A6- 8D B0 80  1770        STA .6+2
80A9- A5 00     1780        LDA SQZZZZ
80AB- AA        1790 .5     TAX
80AC- F0 16     1800        BEQ .9       ...SPECIAL CASE FOR X=0
                1810 *---STRIKE OUT MULTIPLES---------
80AE- 9D 00 FF  1820 .6     STA $FF00,X  REMEMBER THAT N IS REALLY AT .6+2
80B1- 69 00     1830 .7     ADC #*-*     N = N + INC
80B3- AA        1840        TAX
80B4- 90 F8     1850        BCC .6       DONT'T BOTHER TO ADD, NO CARRY
80B6- 18        1860        CLC
80B7- EE B0 80  1870        INC .6+2      INC HIGH ORDER
80BA- 10 EF     1880        BPL .5       ...NOT FINISHED
                1890 *-------------------------------
80BC- A6 02     1900        LDX START    GET OUR NEXT KNOCKOUT
80BE- E8        1910 .8     INX          POINT AT NEXT ODD NUMBER
80BF- E0 40     1920        CPX #64      UP TO 127
80C1- 90 C8     1930        BCC .2       WE'RE DONE IF X>127
80C3- 60        1940        RTS
                1950 *-------------------------------
80C4- AD B0 80  1960 .9     LDA .6+2
80C7- 8D CC 80  1970        STA .10+2
80CA- 8D 00 FF  1980 .10    STA $FF00
80CD- 8A        1990        TXA
80CE- F0 E1     2000        BEQ .7       ...ALWAYS
                2010 *-------------------------------
```

Sorting and Swapping.....Bob Sander-Cederlof

Jack McDonald, writing in the July 1984 Software News, posed a puzzle for programmers: using nothing more than a series of calls to a SWAP, sort five items into ascending order. SWAP compares two items according to the indexes supplied, and exchanges the items if they are out of order. For example, calls on SWAP which follow the pattern of a "Bubble Sort" would look like this:

```
SWAP (1,2)     SWAP (1,2)     SWAP (1,2)     SWAP (1,2)
SWAP (2,3)     SWAP (2,3)     SWAP (2,3)
SWAP (3,4)     SWAP (3,4)
SWAP (4,5)
```

That is ten swaps, which is more than necessary. You can do it in nine, which was McDonalds Puzzle. He gave an answer, and I found another. It was fun writing some quick code to test various swap-lists.

First I wrote a macro named "S" which loaded the two index numbers into X and Y, and called a subroutine named SWAP. See it in lines 1030-1070.

Then I coded SWAP (lines 1200-1290), which compared two bytes at BASE,X and BASE,Y; if they were out of order, I swapped them around. To make things easy for me, I put BASE at $500, which just happens to be the third line on the video screen. That way I could watch everything happen without struggling to code I/O routines.

I wrote a program which would initialize a 5-byte string to all $01 (no program, really just a data definition at line 1670); another which copies the string to BASE (LOAD, lines 1590-1650); another which counts up from 0101010101 to 0505050505, so that all possible combinations would be run through (NEXT, lines 1770-1870); and another to do all these in connection with SORT, which performed a list of SWAP calls. The result was a method for visualizing and checking various groups of SWAPs to see if they could sort any initial permutation into ascending order. Assemble, and type MGO NEXT to see it all work.

Here is the code, with two possible SWAP orders which work, of nine steps each.

```
1020 *--------------------------------
1030         .MA S
1040         LDX #]1
1050         LDY #]2
1060         JSR SWAP
1070         .EM
1080 *--------------------------------
1090         .MA INC
1100         INC PERM+]1
1110         LDA PERM+]1
1120         CMP #6
1130         BCC :1
1140         LDA #1
1150         STA PERM+]1
1160 :1
1170         .EM
```

---------- APPLE SOFTWARE ----------

NEW!!! FONT DOWNLOADER & EDITOR ($39.00)
Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Can be used with every word processor capable of sending ESC and control codes to the printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed, emphasized, underlined, etc.) apply to custom fonts. Full HIRES screen editor lets you create your own custom characters and special graphics symbols. Compatible with many 'dumb' & 'smart' printer I/F cards. User driver option provided. Specify printer: Apple Dot Matrix Printer, C.Itoh 8510A (Prowriter), Epson FX-80/100 or OkiData 92/93.

DISASM 2.2e - AN INTELLIGENT DISASSEMBLER ($30.00)
Investigate the inner workings of machine language programs. DISASM converts 6502 machine code into meaningful, symbolic source. Creates a standard DOS 3.3 text file which is directly compatible with DOS ToolKit, LISA and S-C (4.0 and MACRO) assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor & Pg Zero pg names included.) An address-based cross reference table provides further insight into the inner workings of machine language programs. DISASM is an invaluable machine language learning aid to both the novice & expert alike. **SOURCE code: $60.00**

S-C ASSEMBLER (Ver4.0 only) SUPPORT UTILITY PACKAGE ($30.00)
• SC.XREF - Generates a GLOBAL LABEL Cross Reference Table for complete documentation of source listings. Formatting control accomodates all printer widths for best hardcopy outputs. • SC.GSR - Global Search and Replace eliminates teadious manual renaming of labels. Search all or part of source. Optional prompting for user verification. • SC.TAB - Tabulates source files into neat, readable form. **SOURCE code: $40.00**

---------- HARDWARE/FIRMWARE ----------

THE 'PERFORMER' CARD ($39.00)
Plugs into any Apple slot to convert your 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu provides easy access to printer fonts. Eliminates need to remember complicated ESC codes and key them in to setup printer. Added features include perforation skip, auto page numbering with date & title. Also includes large HIRES graphics screen dump in normal or inverse plus full page TEXT screen dump. Specify printer: Epson MX 80 with Graftrax-80, MX-100, MX-80/100 with GraftraxPlus, NEC 8092JA, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OpkiData 92/93. Oki bonus: print EMPHASIZED & DOUBLE STRIKE fonts! **SOURCE code: $30.00**

FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM ($25.00)
Communications ROM plugs directly into Novation's Apple-Cat Modem card. Three basic modes: Dumb Terminal, Remote Console & Programmable Modem. Added features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back option and built-in printer buffer. Supports most 80-column displays and the 1 wire shift key mod. Uses a superset of Apple's Comm card and Microsmodem II commands. A-C hardware differences prevent 100% compatibility with Comm card. **SOURCE code: $60.00**

RAM/ROM DEVELOPMENT BOARD ($30.00)
Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip. Use a 6116 type RAM chip for program development or just extra memory. Plug in a preprogrammed 2716 EPROM to keep your favorite routines 'on-line'. A versatile board with many uses! Maps into $Cn00-CnFF and $C800-CFFF memory space. Circuit diagram included.

NEW!!! SINGLE BOARD COMPUTER KIT ($20.00)
Kit includes etched PC board (with solder mask and plated thru holes) and assembly instructions. User provides 6502 CPU, 6116 2K RAM, 6821 dual 8-bit I/O and 2732 4K EPROM plus misc common parts. Originally designed as intelligent printer interface - easily adapted to many applications needing dedicated controller. (Assembled and tested: $119.00)

All assembly language SOURCE code is fully commented & provided in both S-C Assembler & standard EDIT format on an Apple DOS 3.3 diskette. Specify your system configuration with order. Avoid a $3.00 postage and handling charge by enclosing full payment with order (MasterCard & VISA excluded). Ask about our products for the VIC 20 and Commodore 64!

R A K - W A R E
41 Ralph Road West Orange NJ 07052 (201) [illegible]

```
                        1180 *-------------------------------
                        1190 *       SWAP (X,Y)
                        1200 *-------------------------------
0800- BD 00 05          1210 SWAP    LDA BASE,X
0803- D9 00 05          1220         CMP BASE,Y
0806- 90 0B             1230         BCC .1
0808- 48                1240         PHA
0809- B9 00 05          1250         LDA BASE,Y
080C- 9D 00 05          1260         STA BASE,X
080F- 68                1270         PLA
0810- 99 00 05          1280         STA BASE,Y
0813- 60                1290 .1      RTS
                        1300 *-------------------------------
                        1310 *       SORT BY SWAPS
                        1320 *-------------------------------
                        1330 SORT
                        1340         .DO 0     CHANGE TO 1 TO SELECT MCDONALD'S LIST
                        1350         >S 4,5         MCDONALD'S ORDER
                        1360         >S 3,5
                        1370         >S 3,4
                        1380         >S 1,2
                        1390         >S 1,4
                        1400         >S 1,3
                        1410         >S 2,5
                        1420         >S 2,4
                        1430         >S 2,3
                        1440         .ELSE
0814-                   1450         >S 1,4         MY ORDER
081B-                   1460         >S 2,5
0822-                   1470         >S 1,3
0829-                   1480         >S 3,5
0830-                   1490         >S 2,4
0837-                   1500         >S 1,2
083E-                   1510         >S 2,3
0845-                   1520         >S 3,4
084C-                   1530         >S 4,5
                        1540         .FIN
0853- 60                1550         RTS
                        1560 *-------------------------------
0500-                   1570 BASE    .EQ $500
                        1580 *-------------------------------
0854- A2 05             1590 LOAD    LDX #5         COPY PERM LIST TO BASE ON SCREEN
0856- BD 63 08          1600 .1      LDA PERM,X
0859- 9D 00 05          1610         STA BASE,X
085C- 9D 80 05          1620         STA BASE+128,X
085F- CA                1630         DEX
0860- D0 F4             1640         BNE .1
0862- 60                1650         RTS
                        1660 *-------------------------------
0863- 00 01 01
0866- 01 01 01          1670 PERM    .HS 000101010101
                        1680 *-------------------------------
0869- A2 04             1690 CHECK   LDX #4         CHECK IF LIST IS SORTED
086B- BD 01 05          1700 .1      LDA BASE+1,X
086E- DD 00 05          1710         CMP BASE,X
0871- 90 03             1720         BCC .2
0873- CA                1730         DEX
0874- D0 F5             1740         BNE .1
0876- 60                1750 .2      RTS
                        1760 *-------------------------------
0877-                   1770 NEXT    >INC 5         INCREMENT PERM LIST
0886- 90 45             1780         BCC .1         EACH BYTE RANGES FROM
0888-                   1790         >INC 4         01 TO 05
0897- 90 34             1800         BCC .1
0899-                   1810         >INC 3
08A8- 90 23             1820         BCC .1
08AA-                   1830         >INC 2
08B9- 90 12             1840         BCC .1
08BB-                   1850         >INC 1
08CA- 90 01             1860         BCC .1
08CC- 60                1870         RTS            FINISHED
08CD- 20 54 08          1880 .1      JSR LOAD       COPY PERMLIST TO SCREEN
08D0- 20 14 08          1890         JSR SORT       SORT IT ON THE SCREEN
08D3- 20 69 08          1900         JSR CHECK      CHECK IF SORTED
08D6- B0 9F             1910         BCS NEXT       ...SORTED, TRY NEXT SEQUENCE
08D8- 60                1920         RTS            ...NOT SORTED
                        1930 *-------------------------------
```

I also got interested in permutation generation, and came up with the following macros and code to generate all 120 permutations of five items, without any extra steps, each step being the simple interchange of two items. Assemble, and type MGO PERMUTE to see it generate 120 strings of the letters ABCDE in different arrangements.

```
                     1940          .MA SS
                     1950          LDX #]1
                     1960          LDY #]2
                     1970          JSR EXCHANGE
                     1980          .EM
                     1990 *--------------------------------
                     2000 EXCHANGE
08D9- BD 63 08       2010          LDA PERM,X
08DC- 48             2020          PHA
08DD- B9 63 08       2030          LDA PERM,Y
08E0- 9D 63 08       2040          STA PERM,X
08E3- 68             2050          PLA
08E4- 99 63 08       2060          STA PERM,Y
08E7- A2 01          2070          LDX #1
08E9- BD 63 08       2080 .1       LDA PERM,X
08EC- 09 C0          2090          ORA #$C0
08EE- 20 ED FD       2100          JSR $FDED
08F1- E8             2110          INX
08F2- E0 06          2120          CPX #6
08F4- 90 F3          2130          BCC .1
08F6- A9 A0          2140          LDA #$A0
08F8- 20 ED FD       2150          JSR $FDED
08FB- 60             2160          RTS
                     2170 *--------------------------------
                     2180          .MA S3
                     2190   >SS 1,2
                     2200   >SS 1,3
                     2210   >SS 1,2
                     2220   >SS 1,3
                     2230   >SS 1,2
                     2240          .EM
                     2250 *--------------------------------
                     2260          .MA S4
                     2270   >S3
                     2280          JSR $FD8E
                     2290   >SS 1,4
                     2300   >S3
                     2310          JSR $FD8E
                     2320   >SS 2,4
                     2330   >S3
                     2340          JSR $FD8E
                     2350   >SS 3,4
                     2360   >S3
                     2370          JSR $FD8E
                     2380          .EM
                     2390 *--------------------------------
                     2400 PERMUTE
08FC- A2 05          2410          LDX #5
08FE- 8A             2420 .1       TXA
08FF- 9D 63 08       2430          STA PERM,X
0902- CA             2440          DEX
0903- D0 F9          2450          BNE .1
                     2460 *--------------------------------
0905-                2470   >SS 1,1
090C-                2480   >S4
09B9-                2490   >SS 1,5
09C0-                2500   >S4
0A6D-                2510   >SS 1,5
0A74-                2520   >S4
0B21-                2530   >SS 1,5
0B28-                2540   >S4
0BD5-                2550   >SS 1,5
0BDC-                2560   >S4
                     2570 *--------------------------------
0C89- 60             2580          RTS
```

Our //c came in, and we love it. However...

The //c package does not include any DOS 3.3 master. Everything is ProDOS. Of course you do get a DOS 3.3 with most software you purchase. And of course ProDOS includes a disk copier that is supposed to be able to copy DOS 3.3 disks when you need to back up your DOS-based software. However...

The ProDOS disk copier which is being shipped with the //c has a serious bug. When you are copying a DOS-based disk it ignores the volume number recorded on the source disk, and forces the copy to be volume 254. That is fine if the source just happened to be volume 254 also, but chances are it isn't. I have many disks around here which are volume 1. The DOS image and the VTOC both think the disk copied by //c ProDOS is volume 1, but RWTS discovers it is volume 254 and refuses to cooperate any further.

I guess the solution is to use the old faithful COPYA from your DOS 3.3 System Master. Since that doesn't come with a //c system, we are including licensed copies of COPYA and FID on our Macro 1.1 disks now.

More gotchas.... Apple decided it was time to rewrite large chunks of the monitor. Necessarily so, because the disassembler now has to cope with 27 new opcodes and address modes. The removed four entries from the monitor command table, and changed its starting point. This throws off the "$" command in the S-C Macro Assemblers, all versions.

If you have Macro 1.1, the //e version is the one you should be running in your //c. You can fix the "$" command with these patches:

$1000 version	$D000 version	old value	new value
$147B	$D47B	$17	$13
$1486	$D486	$CC	$CD
$148B	$D48B	$15	$11

A more elegant patch is possible, which automatically adjusts for whether you are in a //e or //c. If you want this, and have a 1.1 version prior to serial # 675, send us $5 for an update.

We have tried RAK-Ware's DISASM 2.2e on our //c, and it works fine. It even picks up the 27 new opcodes and address modes automatically, because DISASM links to the monitor disassembler. Older versions of DISASM will not run on a //e or //c.

Orphans and Widows.....Bob Sander-Cederlof

James, a brother of Jesus Christ, wrote: "Pure religion and undefiled before God and the Father is this, to visit the fatherless and widows in their affliction, and to keep himself unspotted from the world." (chapter 1, verse 27, King James Version)

Of course, he was referring to real life and to real people with real needs, but it still serves to introduce this little announcement.

"Orphans" and "widows" are also terms used in word processing to describe the lamentable situation of one line of a paragraph being left all alone on one page, while the rest is on another page. If that one line is the last line of a paragraph which won't quite fit, "she" is forced to the top of the next page, and is a widow. If the lonely line is the first line of a paragraph, dwelling at the bottom of a page, bereft of the rest of its family on the following page, he or she is indeed an orphan.

High class word processors give you the option of automatically "visiting" orphans and widows "in their affliction". Thanks to Bobby Deen, this feature is now (as of June 29th) included in the S-C Word Processor (whether high class or not). When the feature is selected (by the "!orl" directive), orphans get moved to the next page and widows get squeezed onto the current page.

Bobby is also working on, and he says it is now functional but somewhat unfinished, a version that fully uses the 80-column display on the Apple //e. We already had 80-column preview, but he is developing 80-column text display during edit/entry mode.

Speed vs. Space.....Bob Sander-Cederlof

There are always tradeoffs. If you have plenty of memory, you can write faster code. If you have plenty of time, you can write smaller code. In an "academic" situation you may have plenty of both, so you can write "creative" code, stretching the frontiers of knowledge. In a "real" world it seems there is never enough time or memory, so projects have to be finished on a very short schedule, fit in a tiny ROM or RAM, and run like greased lightning.

A case in point is last month's installment of the DP18 series: the SHIFT.MAC.RIGHT.ONE subroutine on page 8 takes about 1827 clock cycles, and fits in 25 bytes. Upon reflection, I see a way to write a 34-byte version that takes only 1029 cycles. If I can use nine more bytes, I can shave about 800 microseconds off each and every multiply. (Maybe a total of a whole minute per day!) That might be important, or it might not; but seeing the two techniques side-by-side is probably valuable.

```
1970 SHIFT.MAC.RIGHT.ONE
1980       LDY #4      4 BITS RIGHT
1990 .0    LDX #1      20 BYTES
2000       LSR MAC
2010 .1    ROR MAC,X
2020       INX         NEXT BYTE
2030       PHP
2040       CPX #20
2050       BCS .2      NO MORE BYTES
2060       PLP
2070       JMP .1
2080 .2    PLP
2090       DEY         NEXT BIT
2100       BNE .0
2110       RTS
```

```
1970 SHIFT.MAC.RIGHT.ONE
1980       LDX #0      FOR X=0 TO 19
1990       TXA         NEW 1ST NYBBLE = 0
2000 .1    STA TEMP    SAVE FOR HI NYBBLE
2010       LDA MAC,X   MOVE LOW NYBBLE
2020       ASL             TO HI SIDE
2030       ASL
2040       ASL
2050       ASL
2060       PHA         SAVE ON STACK
2070       LDA MAC,X   MOVE HI NYBBLE
2080       LSR             TO LOW SIDE
2090       LSR
2100       LSR
2110       LSR
2120       ORA TEMP    MERGE WITH NEW
2130       STA MAC,X       HI NYBBLE
2140       PLA         HI NYBBLE OF NEXT BYTE
2150       INX         NEXT X
2160       CPX #20
2170       BCC .1
2180       RTS
```

The smaller method uses two nested loops. The inner loop shifts all 20 bytes of MAC right one bit. The outer loop does the inner loop four times. If I counted cycles correctly, the time is 4*(19*23+18)+7. The faster method uses one loop to scan through the twenty bytes one time. The timing works out as 20*51+9.

Upon still further reflection, it dawned on me that a 38 byte version could run in 840 cycles! This version processes the bytes from right to left instead of left to right; eliminates the PHA-PLA and STA-ORA TEMP of the second version above; and loops only 19 times rather than 20. The timing is 19*43+23.

```
1970 SHIFT.MAC.RIGHT.ONE
1980       LDX #19        FOR X = 19 TO 1 STEP -1
1990 .1    LDA MAC,X      SHIFT HI- TO LO-
2000       LSR
2010       LSR
2020       LSR
2030       LSR
2040       STA MAC,X      SAVE IN FORM 0X
2050       LDA MAC-1,X    GET LO- OF HIGHER BYTE
2060       ASL
2070       ASL
2080       ASL
2090       ASL
2100       ORA MAC,X      MERGE THE NYBBLES
2110       STA MAC,X
2120       DEX            NEXT X
2130       BNE .1         ...UNTIL 0
2140       LDA MAC        PROCESS HIGHEST BYTE
2150       LSR            INTRODUCE LEADING ZERO
2160       LSR
2170       LSR
2180       LSR
2190       STA MAC
2200       RTS
```

Of course an even faster approach is to emulate the loops I wrote for shifting 10-bytes left or right 4-bits. The program would look like this:

```
1970 SHIFT.MAC.RIGHT.ONE
1980       LDY #4
1990 .1    LSR MAC
2000       LSR MAC+1
             .
             .
             .
2180       LSR MAC+19
2190       DEY
2200       BNE .1
2210       RTS
```

This version takes 2+3*20+4 = 66 bytes. Yet the timing is only (4*6+5)*20+7 = 587 clock cycles. And by writing out the four loops all the way, we use 4*3*20 = 240 bytes; the time would be 4*6*20 or 480 cycles.

How about another example? The MULTIPLY.ARG.BY.N subroutine on the same page last month was nice and short, but very slow. The subroutine is called once for each non-zero digit in the multiplier, or up to 20 times. What it does is add the multiplicand to MAC the number of times corresponding to the current multplier digit. If we assume the distribution of digits is random, with equal probablility for any digit 1...9 in any position, the average number of adds will be 5. Actually there will be zero digits too, so the average will be 4.5 instead of 5, with the subroutine not even being called for zero digits.

For 20 digits, 4.5 addition loops per digit, that is an average of 90 addition loops. And a maximum, when all digits are 9, of 180 addition loops.

Now, if there is enough RAM around, we can pre-calculate all partial products from 1 to 9 of the multiplicand and save them in a buffer area. Each partial product will take 11 bytes. We already have the first one in ARG, so for 2...9 we will need 8*11 or 88 bytes of storage. It will take 8 addition loops to form these partial products. Once they are all stored, the MULTIPLY.ARG.BY.N subroutine will always do exactly one addition loop no matter what the non-zero digit is. Therefore the maximum number of addition loops is 8+20 or 28, compared to 180! And the average (assuming there will be 2 zero digits out of 20 on the average) will be 26 addition loops.

The inner loop in MULTIPLY.ARG.BY.N, called "addition loop" above, takes 20 cycles. If we implement this new method, we will have shortened the average case from 1800 to 520 cycles, and the maximum from 3600 to 560 cycles. Of course the whole DMULT routine includes more time-consuming code, but this subroutine was the biggest factor. Taking the SHIFT.MAC.RIGHT.ONE improvements also, we have shortened the overall time in the average case by 2078 cycles, or 2 milliseconds per multiply. In the maximum case, the savings is nearly 4 milliseconds.

Of course, it takes more code space as well as the 88-byte partial product buffer for the new method. And it will take more time to write such a program. You have to make tradeoffs.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).